Chinese relation classification based on deep learning methods

Zihao Chu, Jiayi Wang, Tiantian Wei

Abstract

Relation classification is an important semantic processing task for which state-of-the-art systems still rely on costly handcrafted features. Deep neural networks (DNNs) have revolutionized the field of natural language processing (NLP). Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), the two main types of DNN architectures, are widely explored to handle various NLP tasks. CNN is supposed to be good at extracting position invariant features and RNN at modeling units of these popular models specifically on relation classification. In this work, we will evaluate, compare and analysis performance of different neural networks developed from CNN or RNN. We will give our overall conclusion for further researches in this track.

KeyWord: Neural Network, Deep Learning, Relation Extraction

1 Introduction

Natural language processing (NLP) has benefited greatly from the resurgence of deep neural networks (DNNs), due to their high performance with less need of engineered features. Relation classification is an important NLP task which is normally used as an intermediate step in many complex NLP applications such as question-answering and automatic knowledge base construction. Since the last decade there has been increasing interest in applying machine learnin approaches to this task (Zhang, 2004; Qian et al., 2009; Rink and Harabagiu, 2010). One reason is the availability of benchmark datasets such as the SemEval-2010, task 8 dataset, which encodes the task of classifying the relationship between two nominals marked in a sentence. For instance, the following sentence contains an example of the Message-Topic relation between the nominals "*thesis*" and "*clinical characteristics*".

This $[thesis]_{e1}$ defines the $[clinical\ characteristics]_{e2}$ of amyloid disease.

There are two main DNN architectures: convolutional neural network (CNN) (LeCun et al., 1998) and recurrent neural network (RNN) (Elman, 1990). In general, CNNs are hierarchical and RNNs sequential architectures. But, how should we choose between them for processing language? For most of the cases, it is tempting to choose a CNN for classification tasks like sentiment classification since sentiment is usually determined by some key phrases, and to choose RNNs for a sequence modeling task like language modeling as it requires flexible modeling of context dependencies. But current NLP literature does not support such a clear conclusion. For example, RNNs perform well on document-level sentiment classification (Tang et al., 2015), and (Dauphin et al., 2016) recently showed that gated CNNs outperform LSTMs on language modeling tasks, even though LSTMs had long been seen as better suited. In summary, there is no consensus on DNN selection for any particular NLP problem. This work compares CNNs, GRUs and LSTMs on relation extraction tasks.

In our project, we experiment 5 different ubiquitous neural networks and compare their performances in relation classification tasks with both English and Chinese corpus. The main contributions of this paper are to verify and compare the advantages of each neural networks and analyze empirically the results and summarize useful conclusions for future studies.

2 Preprocessing

2.1 Chinese Word Segmentation

A big difference between Chinese corpus and English corpus is that English word segmentations are indicated by the blank, while Chinese characters have no sign for the word division. So the first step for Chinese NLP tasks is parsing and obtaining the minimal units of semantic information. In our work, the word segmentation of Chinese medical corpus is realized by the PKUseg toolbox.

2.2 Word Embedding

Word embedding is an important language modeling and feature learning technique in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers. It has been also considered to be among a small number of successful applications of unsupervised learning at present. Currently, among all the word embedding models, Word2Vec is the most popular one, which is the general name of CBOW (Continuous bag-of-words), Skip-gram and GloVe. In this project, we apply GloVe model on our data.

2.2.1 Position indicator

We apply four position indicators(PI) to specify the starting and ending of the nominals: <e1>, </e1>, <e2> and </e2>, which gives an excellent improvement on performance (Zhang et al., 2015).

2.2.2 GloVe

GloVe is designed for encoding meaning as vector offsets in an embedding space. First, the co-occurrence matrix X is a $|V| \times |V|$ matrix where V denotes the dictionary, representing the distribution of co-occurrence between each pair of two words in the corpus. Thus, loss function J is defined as:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^{\mathrm{T}} \widetilde{w}_j + b_i + \widetilde{b}_j - logX_{ij}^2) \tag{1}$$

where w_i and b_i are the word vector and bias respectively of word i, $\widetilde{w}$ and b_j are the context word vector and bias respectively of word j, X_{if} is the number of times word i ooccurs in the context of word j, and f is weighting function. In this project, window size is 5, train epoch is 100 and f is defined as $f(x) = min((x/x_{max})^{\alpha}, 1)$ with $x_{max} = 100$ and $\alpha = 0.75$.

After training, the final word vector for each word i is the sum of w_i and $\widetilde{w}_i$, in order to enlarge robustness of word representations.

3 Neural Networks

At the early stage, Neural Network (NN) mostly appears in form of Multi-Layer Perceptrons (MLP), in which each layer is fully connected and the depth of layer determines its capacity on depicting the realistic. However, with the increase of layers, optimization functions sink more frequently into locally optimal solution, which deviates further from global optimal solution. Meanwhile, MLP faces vanishing gradient due to the *sigmoid* function.

To overcome vanishing gradient, other transmission functions such as *ReLU*, *maxout* etc. replace *sigmoid* and construct the basic form of Deep Neural Network (DNN). Nevertheless, there is no difference between fully connected DNN and MLP in terms of structure, which brings a critical problem, the incredibly large number of parameters. As a result, the model trains data slowly and easily encounters over-fitting. To overcome this challenge, several classical neural networks have been proposed and we will introduce structures of these neural networks that we used as follows.

3.1 Input Layer for all networks

For different neural networks, their first layer of network are the same. Word embeddings transform words into vector representations that capture syntactic and semantic information about the words. So we can represent a senctence from input as $S = \{x_1, x_2, ..., x_n\}$, where n is the length of the input sentence and each word vector $x_i \in \mathbb{R}^{d_w}$ is obtained by fetching from the word embedding matrix $W^{emb} \in \mathbb{R}^{d^w \times V}$.

3.2 Different Hidden Layers

3.2.1 CNN

Convolutional Neural Network (CNN) restricts the number of parameters, focusing on mining local features and preserving position relations among data(Zeng et al., 2014). Therefore CNN has a space depth (while the Recurrent Neural Network (RNN) has a time depth) and as a result, it is most commonly applied in computer vision field. In recent years, CNN has also been applied in other fields, such as natural language processing(Wang et al., 2016).

Convolutional Layer

In CNN, to produce local features around each word in the sentence, we look into a window which switches from the first word to the last word. Denote that t is the window size (t is odd). To ensure that every word vector will be used for the same times, we add $(t-1)/2$ word vectors at the beginning and at the end of the sentence x. Then the window $r_i \in \mathbb{R}^{td_w \times 1}$ is defined as:

$$r_i = (x_{i-(t-1)/2}, ..., x_{i+(t-1)/2})^{\mathrm{T}} \tag{2}$$

The convolutional layer input matrix is the successive concatenation of all the windows: $R = [r_1, r_2, ..., r_n]$. It applies a convolution operation in order to reduce the dimension of the data while extracting enough features. The convolution layer output matrix $R^* \in \mathbb{R}^{m \times n}$ is defined with

$$R^* = f(W_f R + B_f) \tag{3}$$

where $W_f \in \mathbb{R}^{m \times td_w}$ is the filter matrix and m is the number of filters and B_f is a linear bias. Function f is the activate function (non linear), for example, $f = sigmoid()$, $f = ReLU()$ or $f = maxout()$ etc.

Pooling Layer

Pooling layer reduces the dimension of the data by combining the output neurons clusters at one layer with single neuron in the next layer. There are two types of pooling layer: max pooling and average pooling. For most cases, we use max pooling because average pooling might dilute the valuable extracted features. In case of max pooling, the feature vector $z \in \mathbb{R}^{m \times 1}$ is defined with

$$z_i = \max_j (R^*_{i,j}) \tag{4}$$

3.2.2 RNN

Different from other feedforward neural networks which only have connections between layers, RNN has connections between nodes in one layer. Each node is connected with a one-way connection to next node and accepts the state of previous node as its input. Its architecture makes it applicable to tasks which require extracting time-depth features, such as machine translation and speech(Zhang et al., 2015).

Recurrent Layer

In the recurrent layer, for each step t, the network accepts the current word vector x_t and the output of the previous step h_{t-1} as the input, and produces the current output h_t by a linear transform followed by an activation function:

$$h_t = tanh(W_r x_t + U_r h_{t-1} + b_r) \tag{5}$$

where W_r and U_r are weight matrices and b_r is the bias vector.

Since the semantic meaning of a sentence is learned word by word, the feature vector produced by the last step can represent the entire sentence.

3.2.3 LSTM

In the process of training RNNs, the exploding and vanishing gradient problems are often encountered. To solve these problems, a variation of RNN: LSTM was invented. The only difference between RNN and LSTM is that the LSTM unit adds the gate mechanism to control the flow of information(Zhou et al., 2016). A LSTM unit has 3 gates: the input gate controls the extent to which a new value flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. And the activation function of the LSTM gates is often the logistic function.

LSTM Unit

The LSTM unit at t-th word consists of a collection of 6 vectors: an input gate i_t, a forget gate f_t, an output gate o_t, a candidate state u_t, a memory cell c_t and a hidden state h_t. The unit at t-th word receives the current word vector x_t, the previous hidden state h_{t-1}, and the memory cell c_{t-1} to calculate the 6 new vectors:

$$\begin{aligned}
i_t &= f(W_i x_t + V_i h_{t-1} + b_i) \\
f_t &= f(W_f x_t + V_f h_{t-1} + b_f) \\
o_t &= f(W_o x_t + V_o h_{t-1} + b_o) \\
u_t &= tanh(W_u x_t + V_u h_{t-1} + b_u) \\
c_t &= i_t . u_t + f_t . c_{t-1} \\
h_t &= o_t . tanh(c_t)
\end{aligned} \tag{6}$$

In these vectors, W and V are weight matrix, b are bias vectors and f denotes the logistic function.

3.2.4 BiLSTM

One limit of the one-directional LSTM is that each node only uses the information of the words before, but in a sentence, the words after this node are also important to model their semantic information. So the bi-directional LSTM (Zhou et al., 2016)is proposed to make predictions on both the past and the future words. With the bi-directional LSTM, we obtain the prediction of step t by adding the output of the forward LSTM and the backward LSTM: $h_t = h_t^{fw} + h_t^{bw}$.

As mentioned before, in one-directional RNN and LSTM, the sentence-level feature vector is represented by the feature vector produced at the end of the sentence. But in relation learning tasks, because of the long-distance patterns in training data, this representation is not suitable in that it tends to forget long-term information, and it's hard to propagate to early steps in the training stage. So we use the max-pooling method to represent the sentence-level feature, as in CNN: $m_i = max_t(h_t)_i$.

3.2.5 2 Layers CNN

In practice, CNN usually performs better when it has more than 2 convolutional layers. In our work, we choose the 2 layers CNN since it performs the best. After max-pooling of the vectors from the first convolutional layer, we add the second convolutional layer and another max-pooling layer.

3.2.6 Output Layer for all models

The output layer is the same for all the models. We use z to represent the sentence feature vector from different hidden layers. To compute the confidence score of each relation, the feature vector z needs to be transformed and then fed into a softmax classifier to obtain the probability of each relation.

$$o = W_o z \tag{7}$$

where k is the number of possible output relation, and $W_o \in \mathbb{R}^{k \times n}$ is the transformation matrix and $o \in \mathbb{R}^{k \times 1}$ is the final output of the network.

3.2.7 Training

The softmax classifier is one of the basic functions used in the output layer. We define the parameter $\theta = (R, W_f, B_f, W_o)$ to encode all the parameters of the model, and we apply the softmax operation on vector o:

$$p(y = i|x; \theta) = \frac{exp(o_i)}{\sum_{j=1}^{k} exp(o_j)} \tag{8}$$

where y denotes the output relation type and $i \in [1, k]$.

Supposed that we have a training set of N examples $\{x^{(i)}, y^{(i)}; i \in [1, N]\}$, for each set {x,y}, our objective is to maximize the probability of the correct class y, and hence we want to minimize the negative log probability of class y. By averaging over all training sets, we obtain our cross entropy loss function:

$$J(\theta) = -\sum_{i=1}^{N} \log p(y^{(i)}|x^{(i)}; \theta) \tag{9}$$

And in order to avoid overfitting, besides the use of dropout, we also add a L2 regularization term at the end of our loss function:

$$J(\theta) = -\sum_{i=1}^{N} \log p(y^{(i)}|x^{(i)}; \theta) + \lambda \sum_{m} \theta_m^2 \tag{10}$$

To minimize $J(\theta)$ in terms of θ, we use the Adam optimizer (Yin et al., 2017), which is better than SGD optimizer in that it can automatically modify the learning rate, and can use different learning rates on different parameters.

4 Experiments

4.1 Experimental Setup

We evaluate these models using Chinese documents. In the corpus we used, there are 14 directional relations and an additional 'other' relation. We remove all the sentences of 'other' relation because 'other' is an artificial class without clear features. The dataset consists of 17623 training sentences, 4405 testing sentences, and each sentence is annotated with a relation between two given nominals. We selected randomly 4405 sentences from the training set as our development set. For the evaluation of experimental results, we use Macro-F1 on 14 relation types.

We parsed the dataset with PKUSeg as well as obtaining entity types and POS tags attached to words, which assist us in better extracting semantic features from the corpus. We initialized word vectors via GloVe which trained on all our dataset including 'other' class, and we conducted all our experiments with 2 different word embedding dimensions: 300 and 500. We also did the same experiments with word vectors that are trainable or untrainable. We fixed input dimension of the last fully connected layer to 128. In addition, for 2 layers CNN, we set the output channel size of first CNN layer as well as the input channel size of the second layer CNN to 32. For CNNs, we padded all sentences' length to 110 with zeros, and we set window size to 5 for each layer. In order to avoid overfitting, we choose L2 regularization and we add a dropout layer before fully connected layer with the probability of droping out 0.5.

4.2 Experimental Results

The experimental results are presented in the four figures below. The Figure 1 and Figure 2 demonstrate the performances among different input formats with the three models, in which the '+pos tag' means adding word vectors with POS vectors while '+entity type' means transforming all nominal to its entity class.. The Figure 3 shows the performance of each model with word vectors trainable as well as word vectors not trainable. The figure 4 illustrates the performance of the models with embedding dimension of 300 and 500.

4.3 Analysis

As can be seen in the Figure 1, the addition of POS tags can efficiently improve the performance of different neural networks since it adds semantic information of the type of each word. Additionally, the Figure 2 illustrates that word vectors firstly trained by GloVe and then remaining untrainable in the network work better than the word vectors trained together with the network. The Figure 3 indicates that increasing embedding dimension in a proper interval will improve performance. Overall, two layers CNN for embedding dimension 300 along with POS tag receive the best macro-F1 score 75.28.

5 Conclusion

In this paper, we compare the performance of 3 types of classical models on Chinese documents. Consistent with previous work, the performance of basic RNN, LSTM and Bi-directional LSTM on Chinese documents increase in sequence owing to the increase of the structure's complexity. However, our results differs from the previous studies, for CNNs perform the best. It is probably due to CNNs' ability to extract features as much as Bi-LSTM with much less time benefiting from its parallel architecture.

In another aspect, feeding models with pos-tags can improve the performances efficiently, while classifying nominals to its noun classes have only little improvements on the performance but requires a lot of manual work. The increase of word embedding dimensions also shows improvement on the performance but the effect would weaken if we continue to increase our embedding dimensions. Hence, for further study, there is not much space to improve these classical methods. New models with the attention mechanism may be a good direction for further investigation.

References

Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao and Bo Xu. 2016. Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classfication. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 207–212.*

Linlin Wang, Zhu Cao, Gerard de Melo and Zhiyuan Liu. 2016. Relation Classification via Multi-Level Attention CNNs. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 1298–1307.*

Cicero Nogueira dos Santos, Bing Xiang and Bowen Zhou. 2015. Classifying Relations by Ranking with Convolutional Neural Networks. *arXiv:1504.06580v2 [cs.CL] 24 May 2015*

Dongxu Zhang and Dong Wang. 2015. Relation Classification via Recurrent Neural Network. *arXiv:1508.01006v2 [cs.CL] 25 Dec 2015*

Wenpeng Yin, Katharina Kann, Mo Yu and Hinrich Schutze. 2017. Comparative Study of CNN and RNN for Natural Language Processing. *arXiv:1702.01923v1 [cs.CL] 7 Feb 2017*

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou and Jun Zhao. 2014. Relation Classification via Convolutional Deep Neural Network In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 2335–2344.*

Bryan Rink and Sanda Harabagiu. 2010. Classifying semantic relations by combining lexical and semantic resources. In *Proceedings of International Work-*

shop on Semantic Evaluation, pages 256–259.

Longhua Qian, Guodong Zhou, Fang Kong, and Qiaoming Zhu. 2009. Semi-supervised learning for semantic relation classification using stratified sampling strategy. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, pages 1437–1445.*

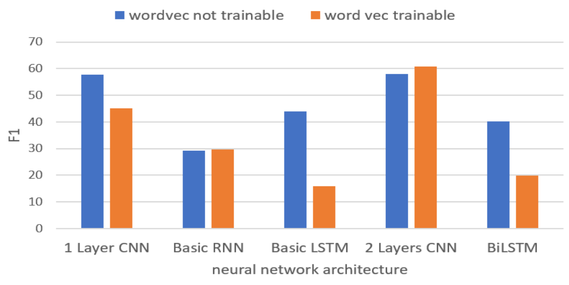
Zhu Zhang. 2004. Weakly-supervised relation classification for information extraction. In *Proceedings of the ACM International Conference on Information and Knowledge Management, pages 581–588, New York, NY, USA.*

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science 14(2):179–211.*

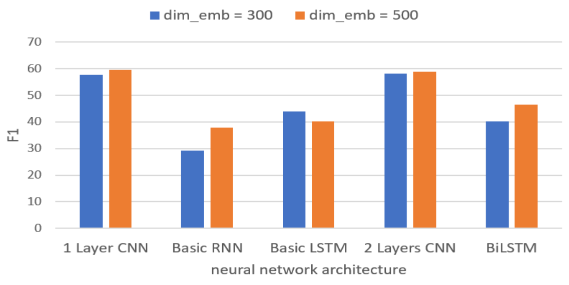
Yann LeCun, L'eon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86(11):2278–2324.*

Duyu Tang, Bing Qin, and Ting Liu. 2015. Document modeling with gated recurrent neural network for sentiment classification. *In Proceedings of EMNLP. pages 1422–1432.*

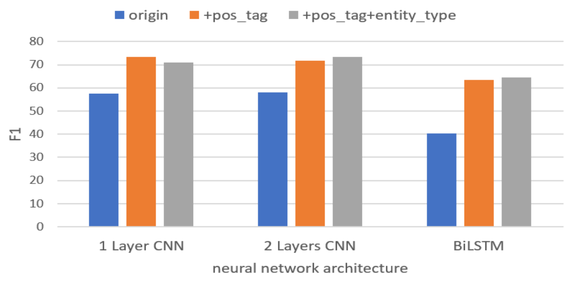
Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language modeling withgated convolutional networks. *arXiv:1612.08083.*



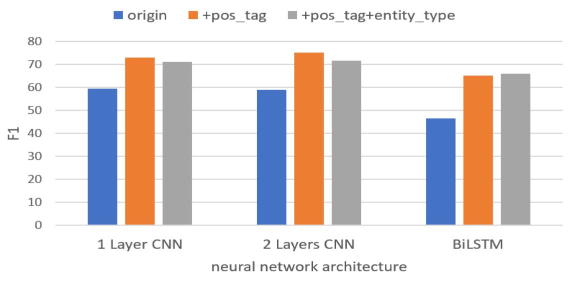
(a) Figure 1: Comparison of different input formats with 3 best models and word embedding dimension 300



(b) Figure 2: Comparison of different input formats with 3 best models and word embedding dimension 500



(c) Figure 3: Comparison of models with word vectors trainable or not



(d) Figure 4: Comparison of models with word embeddings of dimension 300 and 500